



LAWRENCE  
LIVERMORE  
NATIONAL  
LABORATORY

LLNL-TR-440586

# **Level-2 Milestone 3504: Scalable Applications Preparations and Outreach for the Sequoia ID (Dawn)**

**Milestone Report for NNSA HQ**

*Prepared by W. Scott Futral, John Gyllenhaal,  
and Richard Hedges  
June 25, 2010*

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

# Contents

Introduction.....	1
Sequoia Architecture Challenges .....	1
Code Team Outreach Activities .....	2
Applications Testing and Recommendations .....	3
Attachment 1: Milestone Definition Text.....	4
Attachment 2: Handoff Letter.....	5
Attachment 3: The Dawn System Architecture .....	6
Attachment 4: KULL Sequoia ID Port and Initial Performance Characterization .....	7
Attachment 5: I/O Performance Analysis and Tuning for Dawn .....	13

# Introduction

This report documents LLNL SAP project activities in anticipation of the ASC Sequoia system, ASC L2 milestone 3504: Scalable Applications Preparations and Outreach for the Sequoia ID (Dawn), due June 30, 2010. The full text of the milestone is included in Attachment 1. The description of milestone is:

*The SAP effort will develop the knowledge base, documentation, and training to provide ASC code teams with support for utilization of the Sequoia ID (Dawn). SAP will actively engage tri-lab code teams to address their needs in porting codes to the Sequoia ID (Dawn), exploring options for multi-core utilization, characterizing performance issues for the codes. For FY10, tri-lab code teams will be surveyed for needs. One or more multi-physics codes will be engaged to characterize the Sequoia ID (Dawn) performance, analyze bottlenecks and load balance issues, and to develop strategies for improving performance targeting the Sequoia system.*

The milestone was completed on June 25, 2010 when a summary report of porting and performance characterization for the KULL code was presented to the team lead for the code project. The following sections describe the Sequoia system architecture challenges, the project outreach activities, and the applications testing and evaluation done for this milestone.

A letter certifying that the KULL code team was engaged to characterize their code performance on the Sequoia ID system (Dawn) is included as Attachment 2. An overview of the Dawn system hardware is included as Attachment 3. The applications testing details are provided in Attachments 4 and 5.

## Sequoia Architecture Challenges

The Sequoia system scheduled for late CY2011 poses a number of challenges for applications to achieve high performance. In addition to the MPI scalability challenges posed by previous systems, this system will extend scalability requirements to over 1.5 million cores. The individual nodes of the system will feature larger numbers of processors and virtual threads of execution, which will necessitate greater use of programming techniques such as OpenMP or Pthreads to exploit the full capability of the machine. Additional challenges arise from the unique operating system for the BlueGene systems, featuring a Linux microkernel on the compute nodes of the system. The I/O and network infrastructure on Dawn proved to be significant for program startup and

performance of codes in a production setting and will need to be considered in planning by code developers for Sequoia. Attachment 3 provides an overview of the current Dawn system architecture.

## **Code Team Outreach Activities**

The Scalable Applications Preparations (SAP) team developed the knowledge base, documentation, and training to provide ASC code teams with support for utilization of the Sequoia ID system. The following list is incomplete, but it clearly demonstrates the intent of this portion of the milestone requirement was more than satisfied:

- Workshops were held to introduce users to the Dawn system. These were conducted by Blaise Barney, with the assistance of Tom Spelce and Richard Hedges, and were held at the LANL and Sandia sites in early April 2010.
- Interview meetings were conducted with many major code teams over the past year to discuss plans for Dawn and Sequoia and to assess needs of the teams in preparing for those platforms. Meetings with LLNL teams included KULL, Ares, ALE3D, HYDRA, ParaDIS, and DDCMD. Meetings with Sandia teams included Charon (hosted at Livermore), Sierra, Alegra, and CTH (held at Sandia). LANL users Mark Petersen and Daniel Livescu meet with SAP team members to discuss their code requirements for a major calculation targeting Dawn.
- The Dawn User Forum meeting is conducted on the third Thursday of each month and has been very successful in engaging participation from the user community.
- A Web page (<https://computing.llnl.gov/code/sap/>) was established for sharing information and documentation, and as an archive of Dawn User Forum presentations.
- Dawn users are regularly assisted with issues related to compilers, building codes, performance measurement, scalability, memory utilization, and I/O performance.
- Novel Sequoia threading technologies, along with IBM simulators of the Sequoia nodes, are used to evaluate representative code samples from the Sequoia benchmarks, user codes, and newly developed benchmarks in order to gain additional understanding of multi-core node performance issues.
- Significant improvements for performance in code runtimes, I/O, and startup were achieved through close engagement of the SAP team with code teams. These interactions are further detailed in other sections of this report.

## **Applications Testing and Recommendations**

While multiple codes teams were assisted by the SAP effort, particular attention was focused on the KULL project as the multi-physics code to be engaged for characterization of performance on the Sequoia ID (Dawn) system. The resulting report is presented as Attachment 3.

Additionally, extensive and highly successful efforts in addressing I/O performance were realized by the SAP team working with the Ares code team and the SILO/HDF5 team. This work is reported in Attachment 4.

## Attachment 1: Milestone Definition Text

<b>Milestone (ID#):</b> Scalable applications preparations and outreach for the Sequoia ID (Dawn)				
<b>Level:</b> 2		<b>Fiscal Year:</b> FY10		<b>DOE Area/Campaign:</b> ASC
<b>Completion Date:</b> June 30, 2010				
<b>ASC nWBS Subprogram:</b> CSSE				
<b>Participating Sites:</b> LLNL				
<b>Participating Programs/Campaigns:</b> ASC				
<b>Description:</b> The SAP effort will develop the knowledge base, documentation, and training to provide ASC code teams with support for utilization of the Sequoia ID (Dawn). SAP will actively engage tri-lab code teams to address their needs in porting codes to the Sequoia ID (Dawn), exploring options for multi-core utilization, characterizing performance issues for the codes. For FY10, tri-lab code teams will be surveyed for needs. One or more multi-physics codes will be engaged to characterize the Sequoia ID (Dawn) performance, analyze bottlenecks and load balance issues, and to develop strategies for improving performance targeting the Sequoia system.				
<b>Completion Criteria:</b> A report covering the performance findings, and recommended techniques and strategies for the codes studied will be prepared.				
<b>Customer:</b> ASC Integrated Codes Program				
<b>Milestone Certification Method:</b> Professional documentation, such as a report or a set of viewgraphs with a written summary, will be prepared as a record of milestone completion. A "handoff" letter accompanying the report about the project findings and recommendations to the IC program will be document.				
<b>Supporting Resources:</b> ASC the Sequoia ID (Dawn) system				
<b>Supporting Milestones:</b>				
<b>Program</b>		<b>Title</b>		<b>Due Date</b>
N/A		N/A		N/A
<b>Codes/Simulation Tools Employed:</b> The Sequoia ID (Dawn) software environment and IBM Sequoia simulator				
<b>Contribution to the ASC Program:</b> More effective use of Sequoia computer system				
<b>Contribution to Stockpile Stewardship:</b> More effective use of Sequoia computer system				
No.	Risk Description	Risk Assessment (low, medium, high)		
		Consequence	Likelihood	Exposure
1	No access to IBM Sequoia simulator	Low	Medium	Low

## Attachment 2: Handoff Letter



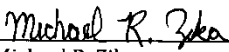
TO: LLNL ASC Office

FROM: Michael Zika

SUBJECT: **Completion of ASC Level 2 Milestone 3504**

I certify that I have received the report "KULL Sequoia ID Port and Initial Performance Characterization", in partial fulfillment of the success criteria for the ASC Level 2 Milestone 3504, "Scalable applications preparations and outreach for the Sequoia ID (Dawn)". The report was received on June 25, 2010, covering the performance findings for the KULL code using the Sequoia ID (Dawn) system and recommending techniques and strategies for improving performance for the Sequoia system.

In addition to the extensive direct engagement with the KULL team on porting to Dawn and addressing performance issues, the Scalable Applications Project (SAP) provided documentation and training opportunities for users of the Dawn and future Sequoia systems. This included on-site visits by IBM experts to present information on Sequoia-related technologies, joint meetings with code teams to discuss options for multi-core utilization, and the monthly Dawn User Forum, among other activities.

  
\_\_\_\_\_  
Michael R. Zika  
Kull Project Leader





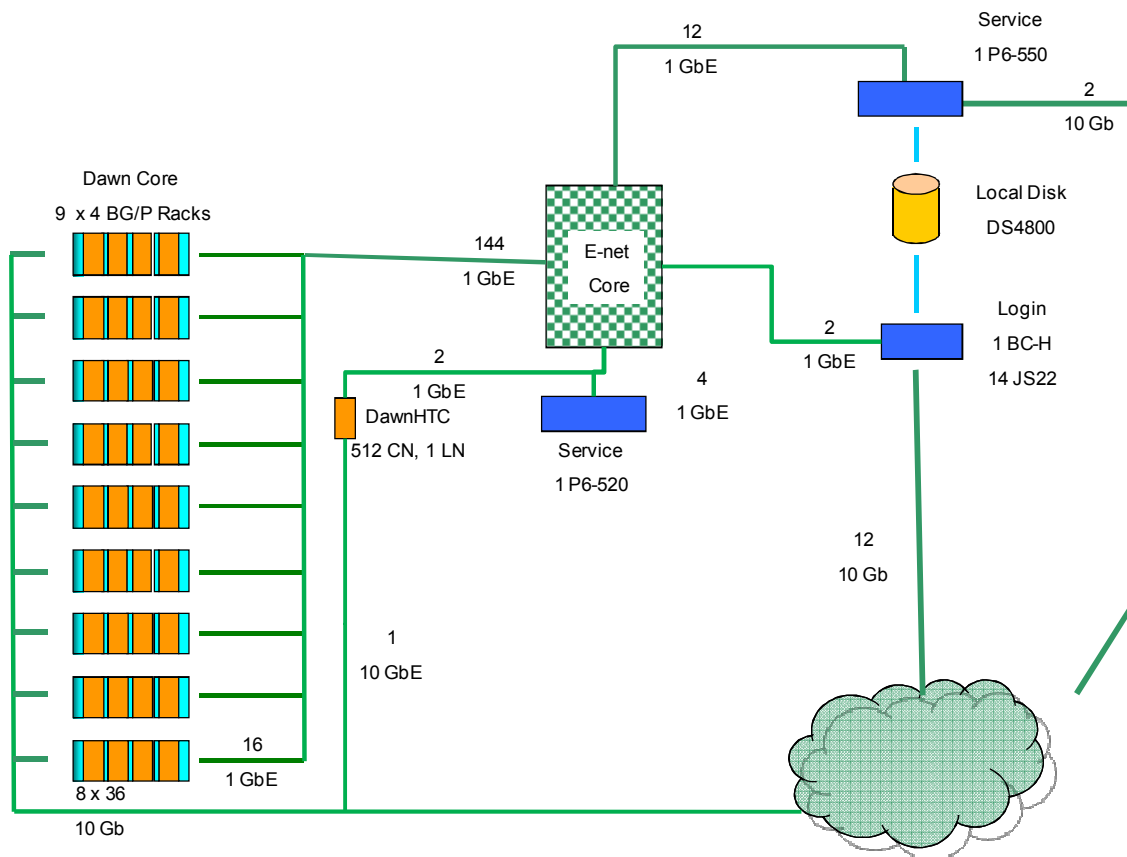
## Attachment 3: The Dawn System Architecture

The Dawn machine is an IBM BlueGene/P system composed of 36 compute racks. Each compute rack has 1024 compute nodes and 8 I/O nodes. Compute nodes and I/O nodes have four PowerPC 450 processor cores and a total of 4 GB of memory. The system has a total of 147,456 compute cores, 288 I/O nodes, 148 TB total memory, and 501 TF peak performance.

End-user access to the system is via the Front End Nodes (FENs). Dawn has 14 FENs, each with four Power6 processors running at 4 GHz with 8 GB of RAM. The FENs are used to compile, run, and monitor jobs, and access file system resources.

Each of the 288 I/O nodes provides I/O services for 128 compute nodes. The I/O nodes mount the NFS and Lustre file systems. The Lustre file system deployed along with Dawn is 2.3 PB in size and delivers a peak of 60 GB/s performance.

Partly as a result of I/O analysis done in this SAP milestone effort, a decision was made to expand the number of I/O nodes to 576. This effort is to be completed in July 2010.



**Dawn System Architecture**

## **Attachment 4: KULL Sequoia ID Port and Initial Performance Characterization**

**(June 2009–June 2010)**

The KULL port to the Sequoia ID system (Dawn, a BG/P system) began in June 2009 with the KULL team's reproduction of IBM's port of Python to Dawn. KULL is layered on top of Python and the Sequoia contract had explicit language about Python support due to porting difficulties on other lightweight kernel systems (like BG/L and RedStorm). IBM provided explicit (and quite counterintuitive) instructions on how to compile Python, and this knowledge was folded into the KULL build system.

After the SAP team provided guidance on best practices for BG/P, the KULL team then spent several weeks porting the third-party libraries their code uses to Dawn. The KULL team found some of the new technology provided to simplify configuring third-party software (the autosubmit feature for serial executables to the HTC cluster) to be unstable. After tracking down and fixing several bugs, the SAP team and LC worked together to implement regular automated testing of these new features so that they could be fixed before the users encountered them.

About this time (July 2009), another large C++ code (Charon, from Sandia) started having serious compiler problems on Dawn, and the KULL team decided to temporarily suspend their porting effort while the SAP team worked to resolve Charon problems. This strategy was successful in that most of the problems we encountered and solved for Charon (by September 2009) were also encountered and quickly solved for KULL when their porting effort resumed in October 2009. KULL very quickly (and successfully) generated an executable using the Charon work-arounds.

The most significant work-around that KULL leveraged from the Charon work is that the GNU linker does not, by default, support executables larger than 16 MB. The GNU linker (ld) is used by both the IBM compiler and the GNU compiler, so both of the available Dawn compilers suffer from the executable size limitation. Charon was the first Dawn executable that encountered this problem (exe size > 70 MB). The SAP team researched this issue and found a well-hidden GNU option (-Wl,--relax) that allowed executables larger than 16 MB to link. Charon was now able to run using the IBM compiler after linking with -Wl,--relax.

Using the same large executable work-around -Wl,--relax, KULL (~900 MB) was now able to compile and successfully link. Unfortunately, the generated KULL executable segmentation faulted before main in C++ class initialization code. Early investigations pointed to a KULL initialization order dependence issue for the problems (and there were thousands of C++ class initializations before main), and three difficult months were spent unsuccessfully trying to discover how to solve the problem. In parallel, we worked on generating a non-export-controlled reproducer for this problem to give to IBM so they

could help with the issue. Our many unsuccessful attempts to reproduce again reinforced that it was likely something KULL was doing to caused the failure.

In January 2010, we changed focus to aggressively cutting code out of KULL until the problem went away in order to discover what source was responsible for the segmentation fault before main. By February 2010, the KULL team found that removing 90% of the code resolved the problem, but it didn't appear to matter which 10% was kept. In mid March, a non-export-controlled reproducer using just four KULL-like global variable declarations linked with 100 MB of automatically generated filler code was produced through the joint effort of the KULL and SAP teams. After analysis by the GNU linker maintainer, it was found in April 2010 to be a bug in GNU ld that randomly corrupted some function call addresses when `-Wl,--relax` was used with executables or shared libraries bigger than 16 MB. All the segfault problems were caused by a GNU linker bug triggered by the large size of the KULL executable and the link mode used to support the use of Python.

The work-around to the GNU linker issue was to limit executable and shared library size to  $< 16\text{MB}$ . KULL is around 900 MB, and the easiest way to meet this requirement was to use the “developer” build target for KULL which generates many (~1000) small shared objects. This developer build target maximizes flexibility in the code development cycle and is regularly used on LLNL linux clusters. The Pynamic benchmark actually was designed to model KULL's shared library usage characteristics with the developer build target. IBM had analyzed Pynamic on Dawn for the Sequoia contract and although IBM had demonstrated it would work, they also found that Dawn was definitely not optimized for this case and strongly recommended against using this build target. Now, this build target appeared to be the only one that would work for KULL. It is worth mentioning that the developer target is actually a desired target for the KULL team if it can be made to work fast on Sequoia (and beyond), because it could allow them to only load the necessary code dynamically, thus dramatically shrinking memory footprint. The current “best” available solution of generating different executables for each mesh type (and not optimizing for the physics used) is both clunky for the users and is not as memory size optimized as a pure shared library version could be made. This feedback has been given to the IBM Sequoia team.

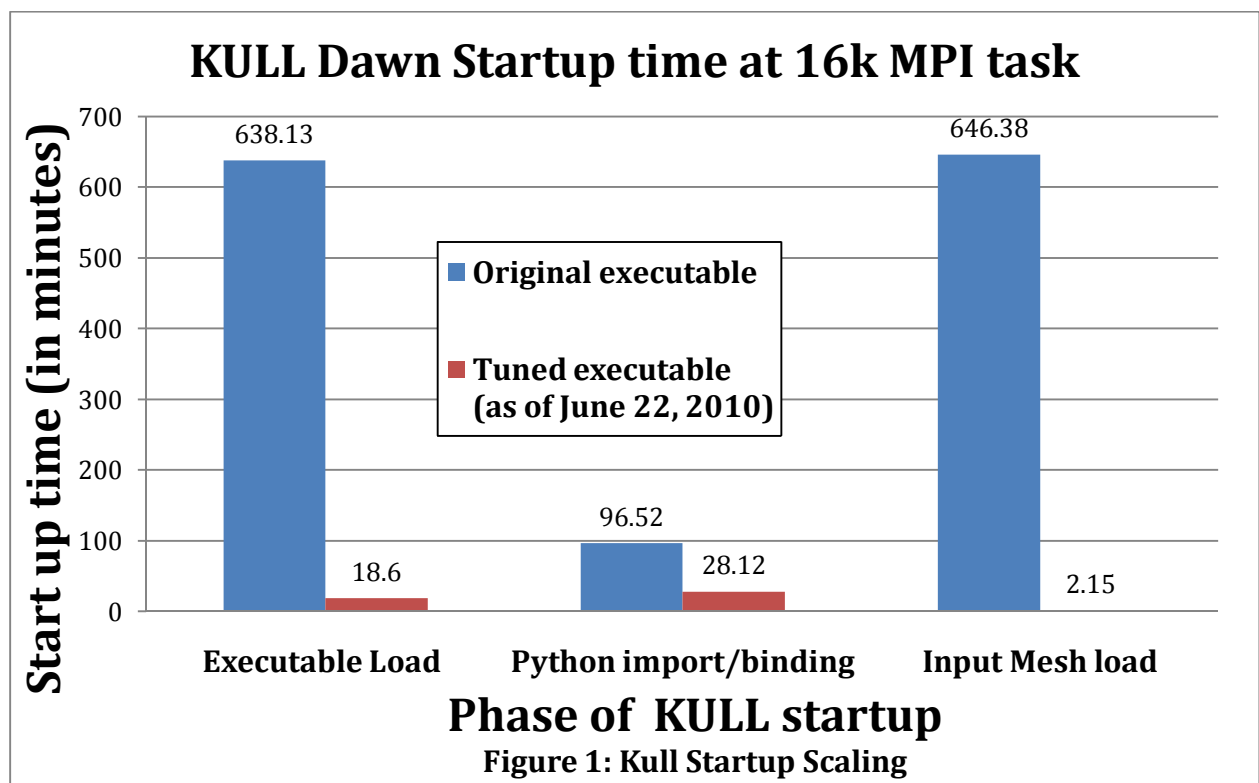
Building and linking KULL with the developer build target resolved the segmentation fault before main issue and allowed the KULL team to start correctness and scalability testing. The regression tests showed that C++ exception handling was broken, and this was traced to another GNU ld issue with a simple work-around. With the C++ exceptions fixed, the KULL team was able to successfully run 600+ regression tests, and the causes of the failing tests were quickly diagnosed.

The KULL team then ran a 2k MPI task multi-physics input (on 2k nodes) that exhibited about a 5X slowdown when compared to the same 2k run on Purple. This 5X slowdown actually met expectations because Purple processors have a ~2.5X faster clock and can issue 5 instructions per cycle (compared to 1 for Dawn). Another multi-physics code also saw the same 5X slowdown at 2k, so KULL's per processor performance on the Dawn processor did not raise red flags (although we continue to look at improving it).

Experiments showed that just loading the debug KULL executable and all those shared libraries was consuming almost 2 GB of memory, thus limiting Dawn runs to 1 task per

4 GB node. At 2k tasks, loading the executable with 1000 shared libraries took a significant amount of time (~ 1 hour), and we became very concerned about load time at scale.

The KULL team then ran a 16k MPI task input on 16k nodes (1 MPI task per node), and the initial serious scaling issues on Dawn became evident. The executable load time (with the 1000 shared libraries) jumped to 10.6 hours, just to get to main(). Importing KULL into Python took an additional 1.6 hours. Simply loading a 700 KB mesh file into each task took over 10 hours (before the job was killed, so it probably would have taken longer). These initial startup times at 16k MPI tasks are shown in Figure 1 under the “Original executable,” and the most recent “Tuned executable” performance numbers (as of June 22, 2010) are also shown for comparison. The SAP team worked with the Dawn system administrator during this run to obtain diagnostics to determine the cause of this huge startup slowdown.

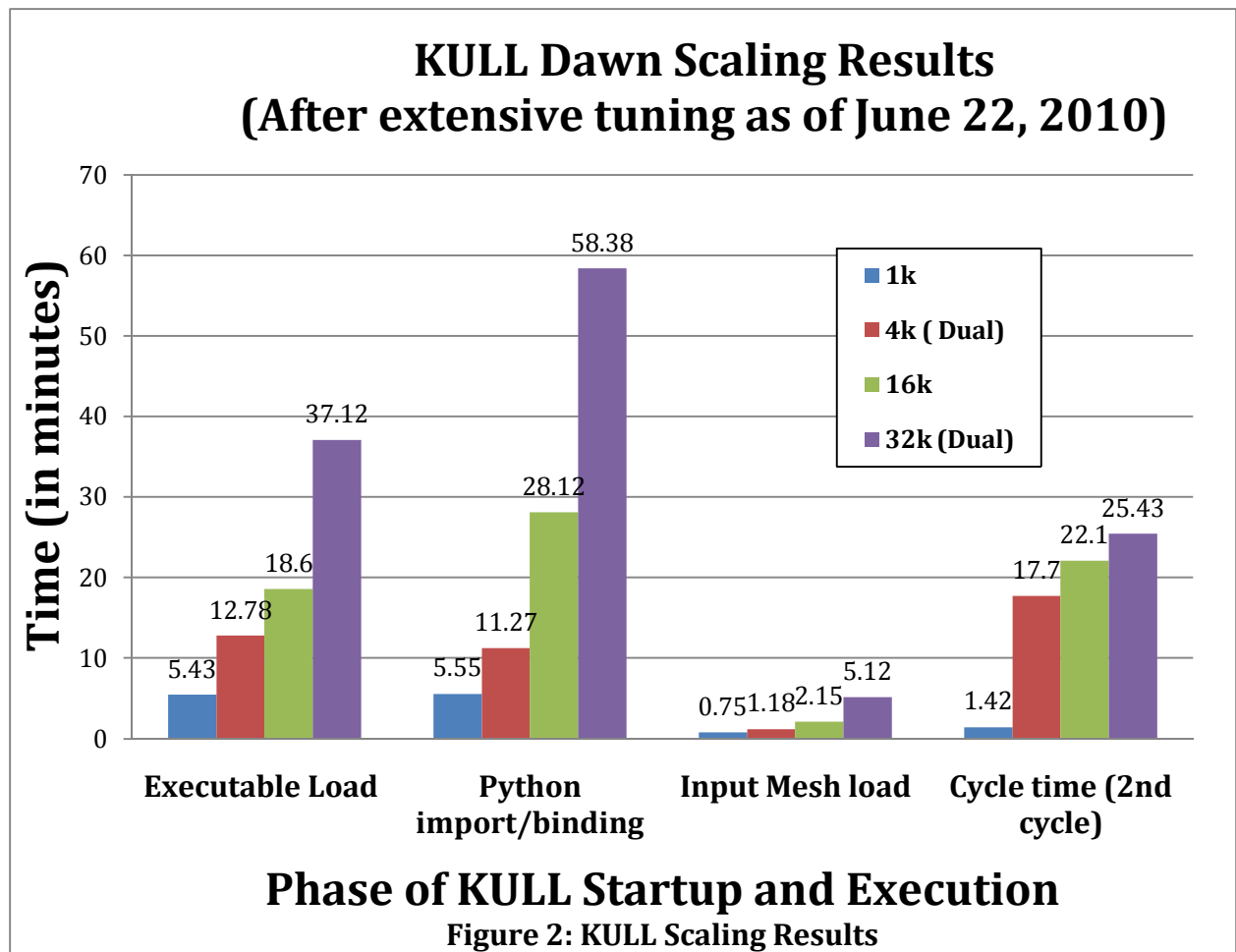


The primary cause of the 10.6 hour executable load time was the startup code making about 300,000,000 unnecessary open() calls to a weak, untuned NFS server (used primarily for Dawn’s compiler and MPI libraries). The POSIX standard prohibits caching of open() calls to non-existent files (because it could have been recently created); thus, the I/O nodes on Dawn had to pass every call to the NFS server. The default shared library search path as defined by the compiler effectively made the startup code always look in 20 places for each of the 1000 shared libraries that didn’t exist before looking in the proper place.

The SAP team, working with the KULL team, devised a set of scripts to pack the 1000 shared libraries into ~ 40 shared libraries, each still less than 16 MB. Only about 300 MB of the shared libraries were actually required for this specific input, so we also eliminated

the other 600 MB of shared libraries. Lastly, the compiler link line was carefully constructed so that only one directory, holding all the required shared libraries, was searched by the startup code. This single shared library directory was put on a powerful NFS system with about 10X the bandwidth of the originally file server. With the current 16 MB constraint (due to the GNU ld bug still being worked by IBM), the executable load has been made as optimal as possible without eliminating unused physics code for this particular input.

The optimized executable load now takes 18.6 minutes at 16k tasks (versus 10.6 hours) and 37.1 minutes at 32k tasks (Figure 2). It is hoped that this time can be further reduced once the GNU ld bug is fixed and most of the KULL code can be put into the main executable. BG/P has an optimized executable launching mechanism that is not currently being well utilized because almost all the KULL code is in shared libraries.



The long Python import time of 1.6 hours is harder to reduce. It is also a problem caused by about 65,000,000 unnecessary open calls as Python searches for various .py, .pyc, .so, etc., files during the import. (It just happened to slam a much more powerful NFS server than the executable load.) The SAP team looked at Python options and found a way for many of the Python files to be combined into a “zip” file. This use of the zip file removed only a quarter of the unnecessary open calls but reduced the import time by more than 3X to 28.1 minutes at 16k (see Figure 1). At 32k MPI tasks, the Python import time

increased to 58.4 minutes (Figure 2). Further reducing the Python import time is still being actively pursued, but its solution may require fundamentally changing the way Python import works, which is a fairly large code development effort. The goal is to drastically reduce the remaining unnecessary open calls and the Python import/bind time to just a few minutes.

The mesh load time ( $> 10.6$  hours at 16k) was due to the underlying I/O library performing a large number of unnecessary lseek on the file and very short reads. The SAP team worked with the I/O library writer, providing guidance on tools (like a Dawn-specific compute-node strace) to see the effect of I/O library changes. After several I/O library iterations, the mesh load time at 16k dropped from more than 10.6 hours to 2.2 minutes (Figure 1). At 32k MPI tasks, reading the mesh took just 5.1 minutes (Figure 2). It is believed that the mesh load time is now near optimal with the current Dawn I/O infrastructure. Due to the SAP teams analysis of I/O on Dawn even for tuned I/O, the decision was made to purchase hardware to double the I/O bandwidth for Dawn. The I/O server expansion is in progress with estimated availability in July 2010.

Figure 2 summarizes the scaling results described below for the most recent, extensively tuned, KULL executable as of June 22, 2010. The 1k and 16k MPI task results were run with 1 MPI task per node. The 4k and 32k MPI task results were run 2 MPI tasks per node (dual mode) because it is much easier to get 2k and 16k node allocations at this time due to the way Dawn is configured. Dual mode does perturb the timings somewhat because of more I/O and MPI switch contention, but the results are representative of what was seen with earlier tuned versions when using 32k nodes for 32k MPI tasks.

After the KULL startup time at 16k MPI tasks on Dawn was reduced from more than 24 hours to around 49 minutes (as of June 22, 2010), simulation cycle time tuning began. Although KULL simulation cycle time at 2k was reasonable (1.4 minutes a cycle), above 2k MPI tasks the cycle time was not acceptable (taking hours per cycle). Using the TAU toolset, the primary non-scaling component was found to be the solver, and further investigation by the solver team found it to be convergence problems. The solver iteration time was scaling well, but the number of iterations ballooned above 2k tasks for the input used, causing the 25.43 cycle time at 32k MPI tasks shown in Figure 2. At the time of this report (June 25, 2010), the solver scaling issues were still being worked, as well as continuing effort to improve performance of all aspects of KULL on Dawn.

There were several exascale lessons learned from the initial KULL BG/P porting and tuning effort. As with past scaling efforts, approaches that were acceptable at the old scale (2k MPI tasks) are not acceptable at the new scale (32k). In this case specifically, open() calls to non-existent files were found to be too expensive at 16k MPI tasks. These open() calls to non-existent files were not directly from the application but from the system startup code handling KULL's shared libraries and from the Python scripting language layer on by KULL. The approach used by both the startup code and Python has the advantage of being straightforward to implement and understand, with the disadvantage that it does not scale. A scalable solution appears doable but requires a significantly more sophisticated implementation to provide the same functionality at scale to the user. This sophistication is worth implementing so that every application does not have to work around the same problems at scale.

Another exascale lesson learned was that most I/O libraries will have to be extensively tuned to read and write large chunks of data at a time, likely via aggressive internal caching, in order to function at extreme scale. It will no longer be possible to keep doing I/O the way the application has always done it and have it work acceptably. Those applications that don't tune their I/O will most likely have both an extremely poor experience at scale and have a significant negative impact on other applications that are sharing the file systems with them.

## **Attachment 5: I/O Performance Analysis and Tuning for Dawn**

This analysis is the result of discussions with several of the ASC code teams on I/O related issues. There were formal meetings with the Ares, KULL, and HYDRA teams where I/O issues were specifically addressed and informal discussions were held touching upon the ALE3D code.

The codes tend to have a similar architecture with regard to the I/O: each depends upon I/O middleware (e.g., SILO, HDF5, PDB) so that the resulting output files are self-describing. One aspect of the self-describing file approach is that the middleware has its own metadata to manage, which leads to small I/O transfers that may be inserted in non-contiguous portions of the file.

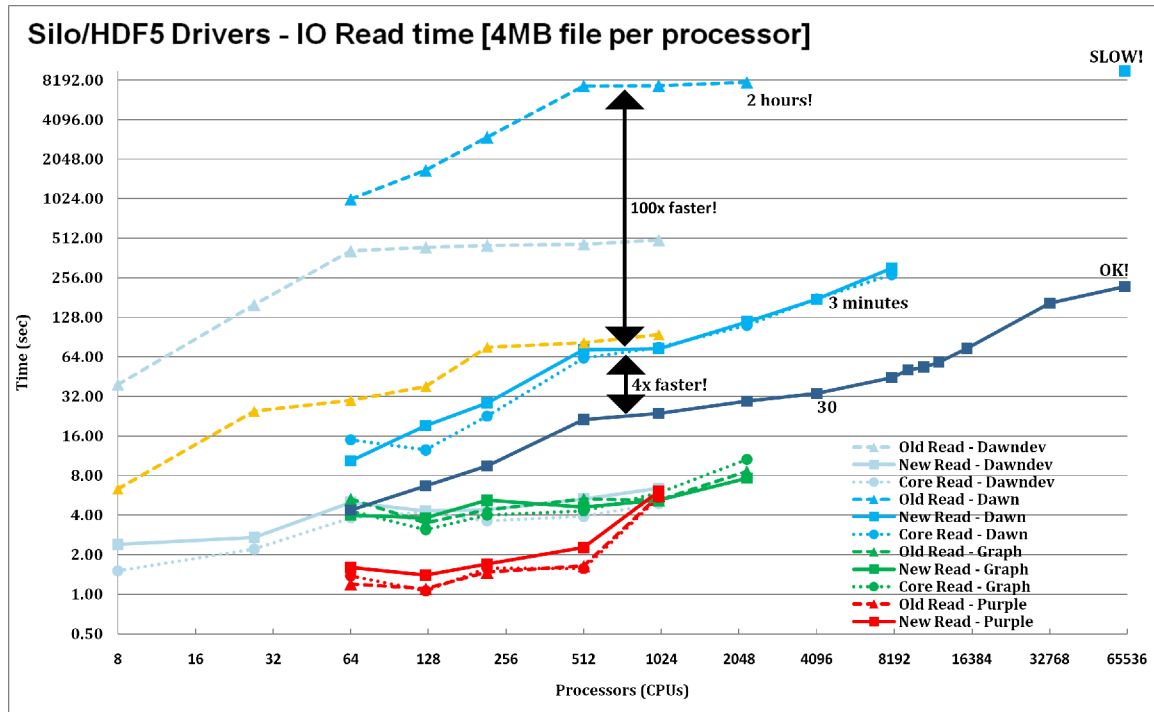
A number of the codes were profiled to assess distribution of transfer sizes. As feared, the number of small transfers to manage the metadata dominated the set of I/Os. In one representative case, 80% of transfers were under 100 bytes.

Concurrently with these meetings and analyses came the observations from the code teams that the I/O performance had become troublesome. This, in combination with the results of the analysis led us to conclude that the I/O nodes of the Sequoia system had become a severe bottleneck. In particular, the client side caching for the Lustre file system, which is so effective on the TLCC Linux clusters, could not be handled by a relatively weak I/O node on the Sequoia ID system, where it may need to be managing up to 512 I/O streams instead of the one to eight in the TLCC case.

We approached the issues from both hardware and software perspectives. For hardware, we doubled the I/O node count on the system. For software, we systematically worked with the code teams so that applications would present a more optimal set of I/O transfers to the file system.

An initial test was performed for the Ares code, for example, where the file was written in its entirety into the compute nodes physical memory, then was flushed to the file system. This general approach was enhanced for the Ares code by Mark Miller. He implemented a new custom HDF5 virtual file driver for SILO which presents large I/O transfers to the file system while conserving compute node memory. This led to a speedup for the I/O of roughly 50X for Ares. It is expected that all of these code improvements will apply directly to the ALE3D code. Further, analogous enhancements, possible for the Kull and Hydra codes, are being pursued. Aggregate improvement in I/O performance results for Ares are illustrated in the following figure.





**Ares I/O Improvements Demonstrated on Sequoia I.D. System**

From this evaluation there evolved a general optimization strategy applicable to the Sequoia ID system and to any future systems where I/O calls are forwarded from a compute node to a node dedicated to I/O and perhaps other system functions. This architecture description applies not only to Sequoia but to all anticipated future HPC systems at Livermore Computing. This strategy is succinctly stated as follows: assemble large contiguous regions of the file in memory on the compute node and forward large contiguous and aligned portions of the file to the I/O nodes. While there may be multiple ways to realize this strategy, experience has demonstrated it will be a requirement in any optimal I/O implementation.